Audit

Greener coin

Date : April 21th, 2023

Summary

I. Introduction

1. About Black Paper

Black Paper has been created to help developer teams. Our goal is to help you to make your smart contract safer.
Cybersecurity requires specific expertise which is very different from smart contract development logic. To ensure everything is well fixed, we stay available to help you.

2. Methodology

a. Preparation

This smart contract is highly standard. It adheres to the specifications outlined in the OpenZeppelin libraries.

b. Audit

Before manually auditing, we pass contracts into automatic tools. This allows us to find some easy-to-find vulnerabilities.
Afterward, we manually go deeper. Every variable and function in the scope are analyzed.

You can find many articles on the lesson website. Here is a snippet list of what we will test :

- Constructor Mismatch
- Ownership Takeover
- Redundant Fallback Function
- Overflows & Underflows
- Reentrancy
- Money-Giving Bug
- Blackhole
- Unauthorized Self-Destruct
- Revert DoS
- Unchecked External Call
- Gasless Send
- Send Instead Of Transfer
- Costly Loop
- (Unsafe) Use Of Untrusted Libraries
- (Unsafe) Use Of Predictable Variables
- Transaction Ordering Dependence
- Deprecated Uses
- Proxy function selector clash

We focus on logic execution exploits, and help to optimize gas price.

c. Reporting

Every point in the code is subject to internal discussions with the team. At this stage, a majority of the probable issues have already been identified and documented.

Post the completion of the code review, analysis, and testing, we prepare a report which contains for each vulnerability :

- Explanation
- Severity score
- How to fix it

Here are severity score definitions.

Critical	A critical vulnerability is a severe issue that can cause significant damage to the contract and its users. These vulnerabilities are easy to exploit and can result in the loss of funds, theft of sensitive data, or other serious consequences. Immediate attention is required to address these vulnerabilities.
Major	A major vulnerability is an issue that can cause significant problems for the contract and its users, but not to the same extent as a critical vulnerability. These vulnerabilities are also easy to exploit and may result in the loss of funds or other negative consequences, but they can be mitigated with timely action.
Medium	A medium vulnerability is an issue that could potentially cause problems for the contract and its users, but the difficulty to exploit is higher than major or critical vulnerabilities. These vulnerabilities may pose a risk to the contract's functionality or security, but they can be addressed without causing significant disruption.
Low	A low vulnerability is a minor issue that does not pose a significant risk to the contract or its users. These vulnerabilities are difficult to exploit and may be cosmetic or technical in nature, but they do not compromise the contract's security or functionality.
Informational	An informational finding is not a vulnerability but rather a suggestion or recommendation for improvement. These findings may include best practices for contract design, suggestions for improving code readability, or other non-critical issues. While not urgent, addressing these findings can help to optimize the contract's performance and reduce the risk of future vulnerabilities.

3. Disclaimer

In this audit, we sent all vulnerabilties found by our team. **We can't guarantee all vulnerabilities have been found.**

4. Scope

There is one smart contract which is stored in the Polygon blockchain at 0x36eC6622B5227e3A7E4d085f1dfc1dC8F5192289.

II. Vulnerabilities

1. Overview

Critical

No critical issues were found.

Major

No major issues were found.

Medium

1 medium issue was found :

- MED-1 Centralization risk in owner keys

Low

No low issues were found.

Informational

3 informationals issues were found :

- INF-1 Dead-code
- INF-2 Floating pragma version
- INF-3 No import statements are used

2. Findings

MED-1 Centralization risk in owner account

Impact: Medium

The owner role is assigned to a single externally owned account (EOA). It can lead to centralization and an increased risk of private key leaks.

Recommendation: To mitigate this risk, we recommend replacing the EOA with a multisignature wallet that is jointly owned by multiple individuals. This would distribute control and reduce the likelihood of a single point of failure.

INF-1 Dead-code

Impact: Informational

_msgData() is never used.

Recommendation: It can be removed to help code readability.

INF-2 Floating pragma version

Impact: Informational

Pragma version is not fixed. Consider locking the version pragma to the same Solidity version used during development and testing. Also consider setting this version to be the latest release.

Recommendation: It is always recommended that pragma should be fixed to the version that you are intending to deploy your contracts with. Replace the pragma version (line 6) by :

```
pragma solidity 0.8.17;
```

Since the contract is already in production, there is no need to take any action.

INF-3 No import statements are used

Impact: Informational

The import of code helps an external developer making it easier to understand and digest the entire Solidity codebase of your project.

Recommendation: Replace all OpenZeppelin contracts by :

```
import "@openzeppelin/contracts/token/ERC20/ERC20.sol";
import "@openzeppelin/contracts/token/ERC20/extensions/ERC20Burnable.sol";
import "@openzeppelin/contracts/security/Pausable.sol";
import "@openzeppelin/contracts/access/Ownable.sol";
```